Faster Polynomial Multiplication via Discrete Fourier Transforms

Alexey Pospelov*
Computer Science Department, Saarland University
pospelov@cs.uni-saarland.de

October 7, 2010

Abstract

We study the complexity of polynomial multiplication over arbitrary fields. We present a unified approach that generalizes all known asymptotically fastest algorithms for this problem. In particular, the well-known algorithm for multiplication of polynomials over fields supporting DFTs of large smooth orders, Schönhage-Strassen's algorithm over arbitrary fields of characteristic different from 2, Schönhage's algorithm over fields of characteristic 2, and Cantor-Kaltofen's algorithm over arbitrary algebras—all appear to be instances of this approach. We also obtain faster algorithms for polynomial multiplication over certain fields which do not support DFTs of large smooth orders.

We prove that the Schönhage-Strassen's upper bound cannot be improved further over the field of rational numbers if we consider only algorithms based on consecutive applications of DFT, as all known fastest algorithms are. We also explore the ways to transfer the recent Fürer's algorithm for integer multiplication to the problem of polynomial multiplication over arbitrary fields of positive characteristic.

This work is inspired by the recent improvement for the closely related problem of complexity of integer multiplication by Fürer and its consequent modular arithmetic treatment due to De, Kurur, Saha, and Saptharishi. We explore the barriers in transferring the techniques for solutions of one problem to a solution of the other.

1 Introduction

Complexity of polynomial multiplication is one of the central problems in computer algebra and algebraic complexity theory. Given two univariate polynomi-

*This research is supported by Cluster of Excellence "Multimodal Computing and Interaction" at Saarland University.

als by vectors of their coefficients,

$$a(x) = \sum_{i=0}^{n-1} a_i x^i, \qquad b(x) = \sum_{j=0}^{n-1} b_j x^j, \tag{1}$$

over some field k, the goal is to compute the coefficients of their product

$$c(x) = a(x) \cdot b(x) = \sum_{\ell=0}^{2n-2} c_\ell x^\ell = \sum_{\ell=0}^{2n-2} \sum_{\substack{0 \le i,\, j < n, \\ i+j=\ell}} a_i b_j x^\ell. \tag{2}$$

The direct way by the formulas above requires n^2 multiplications and $(n-1)^2$ additions of elements of k, making the total complexity of the naive algorithm $O(n^2)$. In what follows we call k the *ground field*.

1.1 Model Of Computation

We study the problem of the total *algebraic* complexity of the multiplication of polynomials over *fields*. That is, elements of k are thought of as algebraic entities, and each binary arithmetic operation on these entities has unit cost. This model is rather abstract in the sense, that it counts, for example, an infinite precision multiplication of two reals as a unit cost operation. On the other hand, it has an advantage of being independent of any concrete implementation that may depend on many factors, including human-related, thus it is more universal, see the discussion on this topic in [9, Introduction].

We are concerned with the *total* number of arithmetic operations, i.e. multiplications and additions/subtractions that are sufficient to multiply two degree $n-1$ polynomials. Since the resulting functions can be computed without divisions, it seems natural to consider only *division-free algebraic algorithms*. The inputs of such algorithm are the values $a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1} \in k$, the outputs are the values $c_0, c_1, \ldots, c_{2n-2} \in k$ as defined in (1), (2). Any step of an algorithm is a multiplication, a division, an addition or a subtraction of two values, each being an input, a value, previously computed by the algorithm, or a constant from the ground field. An algorithm computes product of two degree $n-1$ polynomials, if all outputs $c_0, \ldots, c_{2n-2}$ are computed in some of its steps. The number of steps of an algorithm $\mathcal{A}$ is called *algebraic* or *arithmetic* complexity of $\mathcal{A}$.

In what follows, we will always consider division-free algebraic algorithms. A multiplication performed in a step of an algorithm is called *scalar*, if at least one multiplicand is a field constant, and *nonscalar* in the other case. For an algorithm $\mathcal{A}$ which computes the product of two degree $n-1$ polynomials, we define $L^m_{\mathcal{A}}(n)$ to be the number of nonscalar multiplications used in $\mathcal{A}$, and $L^a_{\mathcal{A}}(n)$ to be the total number of additions, subtractions and scalar multiplications in $\mathcal{A}$. We also set $L_{\mathcal{A}}(n) \coloneqq L^m_{\mathcal{A}}(n) + L^a_{\mathcal{A}}(n)$, the *total algebraic complexity* of $\mathcal{A}$ computing the product of two degree $n-1$ polynomials. In what follows,

$\mathbf{A}_k^n$ always stands for the set of division-free algorithms computing the product of two degree $n-1$ polynomials over k,

$$L_k^m(n) \coloneqq \min_{\mathcal{A}\in\mathbf{A}_k^n} L_{\mathcal{A}}^m(n), \quad L_k^a(n) \coloneqq \min_{\mathcal{A}\in\mathbf{A}_k^n} L_{\mathcal{A}}^a(n), \quad L_k(n) \coloneqq \min_{\mathcal{A}\in\mathbf{A}_k^n} L_{\mathcal{A}}(n).$$

When the field k will be clear from the context or insignificant, we will use then the simplified notation: $L^m(n)$, $L^a(n)$ and $L(n)$, respectively. Note, that $L(n)$ needs not to be equal to $L^m(n) + L^a(n)$, since the minimal number of nonscalar multiplications and the minimal number of additive operations and scalar multiplications can be achieved by different algorithms.

1.2 Fast Polynomial Multiplication And Lower Bounds

Design of efficient algorithms and proving lower bounds is a classical problem in algebraic complexity theory that received wide attention in the past. For an exhaustive treatment of the current state of the art we advise the reader to refer to [9, Sections 2.1, 2.2, 2,7, 2.8]. There exists an algorithm $\mathcal{A} \in \mathbf{A}_k^n$, such that

$$L_{\mathcal{A}}^m(n) = O(n), \qquad L_{\mathcal{A}}^a(n) = O(n\log n), \qquad L_{\mathcal{A}}(n) = O(n\log n),^{1} \tag{3}$$

if k supports Discrete Fourier Transformation (DFT) of order 2^l, [9, Chapter 1, Section 2.1] or 3^l, [9, Exercise 2.5] for each $l > 0$. Schönhage-Strassen's algorithm $\mathcal{B} \in \mathbf{A}_k^n$ computes the product of two degree $n-1$ polynomials over an arbitrary field k of characteristic different from 2 with

$$\begin{gathered} L_{\mathcal{B}}^m(n) = O(n\log n), \qquad L_{\mathcal{B}}^a(n) = O(n\log n\log\log n), \\ L_{\mathcal{B}}(n) = O(n\log n\log\log n). \end{gathered} \tag{4}$$

cf. [24], [9, Section 2.2]. In fact, the original algorithm of [24] computes product of two n-bit integers, but it readily transforms into an algorithm for degree $n-1$ polynomial multiplication. For fields of characteristic 2, Schönhage's algorithm [23], [9, Exercise 2.6] has the same upper bounds as in (4). An algorithm $\mathcal{C}'$ for multiplication of polynomials over arbitrary rings with the same upper bound for $L_{\mathcal{C}'}^m(n)$ was first proposed by Kaminski in [17]. However, there was no matching upper bound for $L_{\mathcal{C}'}^a(n)$. Cantor and Kaltofen generalized Schönhage-Strassen's algorithm into an algorithm $\mathcal{C}$ for the problem of multiplication of polynomials over arbitrary algebras (not necessarily commutative, not necessarily associative) achieving the upper bounds (4), see [11].

For the rest of the paper, we will use the introduced notation: $\mathcal{A}$ will always stand for the multiplication algorithm via DFT with complexity upper bounds (3), $\mathcal{B}$ will stand for Schönhage-Strassen's algorithm if $\operatorname{char} k \neq 2$ and for Schönhage's algorithm if $\operatorname{char} k = 2$, both with complexity upper bounds (4), and $\mathcal{C}$ will stand for Cantor-Kaltofen's algorithm for multiplication of polynomials over arbitrary algebras with the same complexity upper bounds as Schönhage-Strassen's algorithm.

[1] In this paper we always use $\log \coloneqq \log_2$.

Upper and lower bounds for $L_k^m(n)$, which is also called the *multiplicative complexity*, received special attention in literature, see, e.g., [9, Section 14.5]. It is interesting, that for each k, there exists always an algorithm $\mathcal{E} \in \mathbf{A}_k^n$ with $L_{\mathcal{E}}^m(n) = O(n)$, if we do not worry that $L_{\mathcal{E}}^a(n)$ will be worse than in (4), see [12, 25].

If $|k| \geq 2n-2$, then it is known, that $L^m(n) = 2n-1$, see [9, Theorem (2.2)]. For the fields k with $n-2 \leq |k| \leq 2n-3$, the exact value for $L_k^m(n) = 3n - \lfloor \frac{|k|}{2} \rfloor - 2$ was proved by Kaminski and Bshouty in [19, Theorem 2] (see [7, Lemma 1] for the proof of the theorem to hold for the multiplicative complexity).

In order to multiply two degree $n-1$ polynomials over $\mathbb{F}_q$ it suffices to pick an irreducible over $\mathbb{F}_q$ polynomial $p(x)$ of degree $2n-1$ and multiply two elements in $\mathbb{F}_q[x]/p(x)$, that is in $\mathbb{F}_{q^{2n-1}}$. Therefore, for finite fields $k = \mathbb{F}_q$ with $|k| = q \leq n-3$, currently best upper bounds for $L_{\mathbb{F}_q}^m(n)$ are derived from Chudnovskys' algorithm for multiplication in finite field extensions [12, 25] and its improvements by Ballet et al. (p stands always for a prime number; in fact all of the following upper bounds hold also for the *bilinear* complexity, which is a special case of multiplicative complexity, when each nonscalar multiplication in an algorithm is of kind $\ell(a_0, \ldots, a_{n-1}) \cdot \ell'(b_0, \ldots, b_{n-1})$ for some linear forms ℓ, $\ell' \in (k^n)^*$):

$$L_{\mathbb{F}_q}^m(n) \leq \begin{cases} 4(1+\frac{1}{\sqrt{q}-3})n + o(n), & q = p^{2\kappa} \geq 25, \text{ [12, Theorem 7.7]}, \\ 4(1+\frac{p}{\sqrt{q}-3})n, & q = p^{2\kappa} \geq 16, \text{ [1, Theorem 3.1]}, \\ 6(1+\frac{4}{q-3})n, & q = p \geq 5, \text{ [3, Theorem 2.3]}, \\ 6(1+\frac{2p}{q-3})n, & q = p^{\kappa} \geq 16, \text{ [2, Theorem 4.6]}, \\ 12(1+\frac{p}{q-3})n, & q > 3, \text{ [1, Corollary 3.1]}, \\ 54n - 27, & q = 3, \text{ [1, Remark after Corollary 3.1]}, \\ \frac{477}{13}n - \frac{108}{13} < 36.7n, & q = 2, \text{ [4, Theorem 3.4]}. \end{cases}$$

The best known lower bounds in case of $k = \mathbb{F}_q$ when $q \leq n-3$ are

$$L_{\mathbb{F}_q}(n) \geq L_{\mathbb{F}_q}^m(n) \geq \begin{cases} \left(3 + \frac{(q-1)^2}{q^5+(q-1)^3}\right) n - o(n), & q \geq 3, \text{ [18]}, \\ 3.52n - o(n), & q = 2, \text{ [6]}. \end{cases}$$

If we allow for a moment divisions to be present in an algorithm, then there is a lower bound $3n - o(n)$ for the total number of nonscalar multiplications *and* divisions necessary for any algebraic algorithm computing product of two degree n polynomials, see [8].

There are few lower bounds for the algebraic complexity of polynomial multiplication. Most of them are actually bounding $L^m(n)$ which can be used as a conservative lower bound for $L(n)$. Since the coefficients $c_0, \ldots, c_{2n-2}$ are linearly independent, in case of division-free algorithms one immediately obtains the lower bound $L(n) \geq L^m(n) \geq 2n-1$ over arbitrary fields. To the moment, this is the only general lower bound for $L(n)$ which does not depend on the ground field. Bürgisser and Lotz in [10] proved the only currently known non-

linear lower bound if $\Omega(n \log n)$ for $L_{\mathbb{C}}(n)$ (actually, on $L^a_{\mathbb{C}}(n)$) which holds in case when all scalar multiplications in an algorithm are with bounded constants.

The gap between the upper and the lower bounds on $L_k(n)$ motivates to look for better multiplication algorithms and for higher lower bounds for the complexity of polynomial multiplication, in particular over small fields. For example, it is still an open problem if the total algebraic complexity of polynomial multiplication is nonlinear, see [9, Problem 2.1]. Another well known challenge is to decrease the upper bound for $L_k(n)$ of (4) to the level of (3) in case of arbitrary fields, see [21] for the more general challenge of multivariate polynomial multiplication. In this paper we partially address both problems.

1.3 Our Results

As our first contribution, for every field k, we present an algorithm $\mathcal{D}_k \in \mathbf{A}^n_k$, which is a generalization of Schönhage-Strassen's construction that works over arbitrary fields and achieves the best known complexity upper bounds. In fact, we argue that the algorithm $\mathcal{D}_k$ stands for a generic polynomial multiplication algorithm that relies on consecutive application of DFT. In particular, the algorithms $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ come as special cases of the algorithm $\mathcal{D}_k$. We are currently not aware of any algorithms with an upper bound of (4) that are not based on consecutive DFT applications and thus do not follow from the algorithm $\mathcal{D}_k$.

As the second contribution, we show that $L_{\mathcal{D}_k}(n) = o(n \log n \log\log n)$ in case when algorithm $\mathcal{A}$ cannot be applied but the field k has some simple algebraic properties that are ignored by algorithms $\mathcal{B}$ and $\mathcal{C}$. This improves the upper bound of (4) over such fields. We also present a parameterization of fields k with respect to the performance of the algorithm $\mathcal{D}_k$, and give explicit upper bounds which depend on this parameterization. More precisely, over each field k, we have $\Omega(n \log n) = L_{\mathcal{D}_k}(n) = O(n \log n \log\log n)$, and over certain fields that do not admit low-overhead application of the algorithm $\mathcal{A}$, the algorithm $\mathcal{D}_k$ achieves intermediate complexities between the indicated bounds.

Finally, we show, that the algorithm $\mathcal{D}_k$ has natural limitations depending on the ground field k. For example, we prove that $L_{\mathcal{D}_{\mathbb{Q}}}(n) = \Omega(n \log n \log\log n)$. Furthermore, we characterize all such fields, where application of DFT-based methods does not lead to any improvement of the upper bound (4). Therefore, we consider this as an exhaustive exploration of performance of generic algorithms for polynomial multiplication based on application of DFT.

1.4 Organization Of the Paper

Section 2 contains the necessary algebraic preliminaries. We then give a uniform treatment of the best known algorithms for polynomial multiplication over arbitrary fields in Section 3: Schönhage-Strassen's algorithm [24], Schönhage's algorithm [23] and Cantor-Kaltofen's algorithm [11]. In Section 4 we remind the best known upper bounds for computation of DFT over different fields and show some efficient applications of their combination. We also indicate limitations of the known techniques.

Section 5 contains our main contributions. We end with one particular number-theoretic conjecture due to Bläser on the existence of special finite field extensions. In fact, if it holds, then the algorithm algorithm $\mathcal{D}_k$ can achieve better performance than that of the previously known algorithms $\mathcal{B}$ and $\mathcal{C}$ over any field of characteristic different from 0.

2 Basic Definitions

In what follows we will denote the ground field by k. *Algebra* will always stand for a finite dimensional associative algebra over some field with unity 1. For a function $f : \mathbb{N} \to \mathbb{R}$, a positive integer n is called *f-smooth*, if each prime divisor of n does not exceed $f(n)$. Note, that this definition is not trivial only if $f(n) < \frac{n}{2}$. If $f(n) = O(1)$, then an f-smooth positive integer is called just *smooth*.

All currently known fastest algorithms for polynomial multiplication over arbitrary fields rely on the possibility to apply the Discrete Fourier Transform by means of the Fast Fourier Transform algorithm (FFT) and on the estimation of the overhead needed to extend the field to make DFTs available. This possibility depends on existence of so-called *principal roots of unity* of large smooth orders, e.g., of orders 2^ν for all $\nu > 0$.

Let A be an algebra over a field k. $\omega \in A$ is called a *principal n-th root of unity* if $\omega^n = 1_A$ (where 1_A is the unity of A) and for $1 \leq \nu < n$, $1 - \omega^\nu$ is not a zero divisor in A. It follows, that if $\omega \in A$ is a principal n-th root of unity, then $\operatorname{char} k \nmid n$ and

$$\sum_{\nu=0}^{n-1} \omega^{i\cdot\nu} = \begin{cases} n, & \text{if } i \equiv 0 \pmod{n}, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

If A is a field, then $\omega \in A$ is a principal n-th root of unity iff ω is a primitive n-th root of unity. For a principal n-th root of unity $\omega \in A$, the map

$$\mathrm{DFT}_n^\omega : A[x]/(x^n - 1) \to A^n$$

defined as $\mathrm{DFT}_n^\omega \left(\sum_{\nu=0}^{n-1} a_\nu x^\nu \right) = (\tilde{a}_0, \ldots, \tilde{a}_{n-1})$, where $\tilde{a}_i = \sum_{\nu=0}^{n-1} \omega^{i\cdot\nu} a_\nu$, for $i = 0, \ldots, n-1$, is called the *Discrete Fourier Transform* of order n over A with respect to the principal n-th root of unity ω.

It follows from Chinese Remainder Theorem that if $\omega \in A$ is a principal n-th root of unity, then DFT_n^ω is an isomorphism between $A[x]/(x^n - 1)$ and A^n. (5) implies that the inverse transform of DFT_n^ω is $\frac{1}{n}\mathrm{DFT}_n^{\omega^{-1}}$ since ω^{-1} is also a principal n-th root of unity in A [9, Theorem (2.6)]: $a_i = \frac{1}{n}\sum_{\nu=0}^{n-1} \omega^{-i\cdot\nu} \cdot \tilde{a}_\nu$, for $i = 0, \ldots, n-1$. Note, that if $\omega \in A$ is a principal n-th root of unity and $a(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1} \in k[x]/(x^n - 1)$, then

$$\mathrm{DFT}_n^\omega \left(a(x)\right) = \left(a(\omega^0), a(\omega), \ldots, a(\omega^{n-1})\right).$$

An important property of the DFT is that it can be computed efficiently under certain conditions, see Section 4. We only mention here, that if $n = s^\nu$ for some constant s, there is a principal n-th root of unity ω in an algebra A, then DFT_n^ω can be computed in $O(n \log n)$ additions of elements of A and multiplications of elements of A with powers of ω.

3 State Of the Art

3.1 Multiplication via DFT

The easiest way to illustrate power of applications of DFT is to consider multiplication of polynomials over a field k which contains primitive roots of unity of large smooth orders. Assume that for some integer constant $s \geq 2$ and for each ν, k contains a primitive s^ν-th root of unity. The well-known DFT-based algorithm $\mathcal{A}$ takes two degree $n-1$ polynomials $a(x)$ and $b(x)$ and proceeds as follows:

Embed and pad Set $\nu = \lceil \log_s(2n-1) \rceil$ such that $s^\nu \geq 2n-1$. Pad the vectors of coefficients of $a(x)$ and $b(x)$ with zeroes and consider $a(x)$ and $b(x)$ as polynomials of degree $s^\nu - 1$ in $k[x]/(x^{s^\nu} - 1)$. This step is performed at no arithmetical cost.

Compute DFTs For a primitive s^ν-th root of unity $\omega \in k$, compute

$$\tilde{a} \coloneqq \mathrm{DFT}_{s^\nu}^\omega(a(x)), \qquad \tilde{b} \coloneqq \mathrm{DFT}_{s^\nu}^\omega(b(x)).$$

The cost of this step is $O(n \log n)$ arithmetical operations over k (recall, that s is a constant).

Multiply vectors Compute dot-product $\tilde{c} \coloneqq \tilde{a} \cdot \tilde{b}$, that is perform $s^\nu = O(n)$ multiplications of elements in k.

Compute inverse DFT Compute

$$\frac{1}{s^\nu} \mathrm{DFT}_{s^\nu}^{\omega^{-1}}(\tilde{c}) = c(x).$$

This step requires $O(n \log n)$ arithmetical operations in k.

As we can see the total complexity of $O(n \log n)$ arithmetic operations over k. Note, that the number of multiplications is $s^\nu \leq 2ns - s$, and is linear in n as long as s is a constant.

3.2 Multiplication in Arbitrary Fields

Now suppose that k does not contain the needed primitive roots of unity. The methods we will describe now are all based on the idea of an algebraic extension $K \supset k$ where the DFT of a large smooth order s^ν is defined. In these methods one encodes the input polynomials into polynomials of smaller degree

over K and uses the algorithm $\mathcal{A}$ over K to multiply these polynomials. The s^ν multiplications of elements in K are performed via an efficient reduction to multiplication of polynomials of smaller degree, thus making the whole scheme recursive.

3.2.1 Schönhage-Strassen's Algorithm

Assume that $\operatorname{char} k \neq 2$. In this case, x is a $2n$-th principal root of unity in $A_n := k[x]/(x^n+1)$, which is a k-algebra of dimension $n \dim A$ [9, (2.11)] and $A[x]/(x^n+1) \cong A[x]/(x^n-1)$, if a k-algebra A contains a principal $2n$-th root of unity [9, (2.12)]. For $n \geq 3$, Schönhage-Strassen's algorithm [24], which we denote by $\mathcal{B}$ takes two degree $n-1$ polynomials $a(x)$ and $b(x)$ over k and proceeds as follows:

Embed and pad Set $\nu = \lceil \log_2(2n-1) \rceil \geq 2$ such that $N := 2^\nu \geq 2n-1$. Pad the vectors of coefficients of $a(x)$ and $b(x)$ with zeroes and consider $a(x)$ and $b(x)$ as polynomials of degree $N-1$ in A_N. This step is performed at no arithmetical cost.

Extend Set $N_1 := 2^{\lceil \frac{\nu}{2} \rceil} \geq 2$, $N_2 := 2^{\lfloor \frac{\nu}{2} \rfloor + 1}$, such that $\frac{N_1}{2} \cdot N_2 = N$. Encode $a(x)$ and $b(x)$ (considered as elements of A_N) as polynomials of degree $N_2 - 1$ over $A_{N_1} = k[y]/(y^{N_1}+1)$:

$$a(x) = \sum_{i=0}^{N-1} a_i x^i \mapsto \sum_{i=0}^{N_2-1} \underbrace{\left(\sum_{j=0}^{\frac{N_1}{2}-1} a_{\frac{N_1}{2} \cdot i + j} y^j \right)}_{=: \bar{a}_i \in A_{N_1}} \underbrace{\left(x^{\frac{N_1}{2}}\right)}_{\bar{x}}{}^i = \sum_{i=0}^{N_2-1} \bar{a}_i \bar{x}^i =: \bar{a}(\bar{x}).$$

y is a $2N_1$-th principal root of unity in A_{N_1} and $2N_1 \geq N_2$, all powers of 2. Since $N_2 \mid 2N_1$, $\psi := y^{\frac{2N_1}{N_2}}$ is a principal N_2-th root of unity in A_{N_1}.

Compute DFTs of orders N_2 of $\bar{a}(\bar{x})$ and $\bar{b}(\bar{x})$ with respect to ψ. Note, that addition of two elements in A_{N_1} can be performed in N_1 additions in A, and multiplication by powers of ψ, that is, by powers of y results in cyclic shifts and sign changes and is also bounded by N_1 additions (if we count a sign change as an additive operation). Therefore, this step requires $O(N_1 \cdot N_2 \log N_2) = O(N \log N)$ arithmetic operations over k.

Multiply the coordinates of $\tilde{\bar{a}} \cdot \tilde{\bar{b}} = \tilde{\bar{c}}$. This results in computing N_2 products of polynomials of degree $\frac{N_1}{2} - 1$, which are computed by a recursive application of the currently described procedure.

Compute inverse DFT of $\tilde{\bar{c}}$ with respect to $\psi^{-1} = y^{2N_1 - \frac{2N_1}{N_2}}$. As before, this requires $O(N \log N)$ additive operations in k.

Unembedding in this case is can be computed in the following way: since degrees in y of all coefficients $\bar{a}_i$, $\bar{b}_i$ were at most $\frac{N_1}{2} - 1$, and they were

multiplied in A_{N_1}, degrees in y of all coefficients $\bar{c}_i$ are at most $N_1-2 < N_1$. Therefore, for all $i = 0, \ldots, N_2 - 1$,

$$\bar{c}_i = \sum_{j=0}^{N_1-1} c_{i,\,j} y^j$$

are already computed with some $c_{i,\,j} \in k$, and

$$c(x) = \sum_{i=0}^{N_2-1} \bar{c}_i (x^{\frac{N_1}{2}})^i = \sum_{i=0}^{N_2-1} \sum_{j=0}^{N_1-1} c_{i,\,j} x^{\frac{N_1}{2} \cdot i + j}$$

$$= \sum_{i=0}^{N-1} (c_{\lfloor \frac{2i}{N_1} \rfloor,\, i - \lfloor \frac{2i}{N_1} \rfloor \cdot \frac{N_1}{2}} + c_{\lfloor \frac{2i}{N_1} \rfloor - 1,\, \frac{N_1}{2} + i - \lfloor \frac{2i}{N_1} \rfloor \cdot \frac{N_1}{2}}) x^i$$

can be computed by at most N additions of elements in k (we assume that $c_{i,\,j} = 0$ if $i < 0$ or $j \geq N_1$).

Denoting by $L'_{\mathcal{B}}(N)$ the total complexity of multiplication in A_N via Schönhage-Strassen's algorithm $\mathcal{B}$, we obtain following complexity inequality:

$$L_{\mathcal{B}}(n) \leq L'_{\mathcal{B}}(N) \leq N_2 L'_{\mathcal{B}}(N_1) + O(N \log N).$$

It implies $L'_{\mathcal{B}}(N) = O(N \log N \log\log N)$ and the desired estimates (4) since $N \leq 4n - 2$. A more careful examination of the numbers of additions and multiplications used gives also the upper bounds (4).

Rough complexity analysis can be also made by following observations. The cost of each recursive step (under a recursive step we understand all the work done on a fixed recursive depth) is $O(N_1 \cdot N_2 \log N_2) = O(n \log n)$ and is defined by the complexity of the DFT used to reduce the multiplication to several multiplications of smaller formats. Note, that in order to adjoin a $2N_1$-th root of unity to k in the initial step we take a (ring) extension of degree N_1, which is a half of the degree of the root we get. This crucial fact reduces the number of recursive steps to $O(\log\log n)$. Thus, the upper bounds (4) for the complexity of $\mathcal{B}$ can also be obtained as a product of the upper bound for the complexity of a recursive step by the number of recursive steps.

3.2.2 Schönhage's Algorithm

Now assume that $\operatorname{char} k = 2$. Again, the first step is the choice of a finite dimensional algebra to reduce the original polynomial multiplication to. In case of $\operatorname{char} k = 2$, the choice of $k[x]/(x^n + 1)$ does not work since it can be used only efficient to append 2^ν-th roots of unity and $x^{2^\nu} - 1 = (x-1)^{2^\nu}$ in every field of characteristic 2. Schönhage's algorithm [23] thus reduces the multiplication of polynomials over k to the multiplication in $B_N := k[x]/(x^{2N} + x^N + 1)$, where x is a $3N$-th principal root of unity. Therefore, we can follow the way of the original Schönhage-Strassen's algorithm with one important modification explained in this section.

For $n \geq 3$, Schönhage's algorithm $\mathcal{B}$ takes two degree $n-1$ polynomials $a(x)$ and $b(x)$ and proceeds as follows:

Embed and pad Set $\nu = \left\lceil \log_3(n - \frac{1}{2}) \right\rceil$ such that for $N \coloneqq 3^\nu$, $2N \geq 2n-1$. Pad the vectors of coefficients of $a(x)$ and $b(x)$ with zeroes and consider $a(x)$ and $b(x)$ as elements of B_N. This step is performed at no arithmetical cost.

Extend Set $N_1 \coloneqq 3^{\lceil \frac{\nu}{2} \rceil}$ and $N_2 \coloneqq 3^{\lfloor \frac{\nu}{2} \rfloor}$ such that $N_1 N_2 = N$. Encode the input polynomials $a(x)$ and $b(x)$ (considered as elements of B_N) as polynomials of degree $2N_2 - 1$ over $B_{N_1} = k[y]/(y^{2N_1} + y^{N_1} + 1)$:

$$a(x) = \sum_{i=0}^{2N-1} a_i x^i \mapsto \sum_{i=0}^{2N_2-1} \underbrace{\left(\sum_{j=0}^{N_1-1} a_{N_1 \cdot i + j} y^j \right)}_{=:\bar{a}_i \in B_{N_1}} \underbrace{\left(x^{N_1}\right)^i}_{\bar{x}} = \sum_{i=0}^{2N_2-1} \bar{a}_i \bar{x}^i =: \bar{a}(\bar{x}).$$

y is a $3N_1$-th principal root of unity in B_{N_1}, and $N_1 \geq N_2$, both powers of 3. Thus, $\psi = y^{\frac{N_1}{N_2}}$ is a $3N_2$-th principal root of unity in B_{N_1}.

Compute DFTs of $\bar{a}(\bar{x})$ and $\bar{b}(\bar{x})$, both padded to degree $3N_2$ with zeroes, with respect to ψ. Note, that addition of two elements in B_{N_1} can be performed in at most $2N_1$ additions of elements in k, and multiplications by powers of ψ, that is, by powers of y can also be performed in $O(N_1)$ operations since $y^{3N_1 i + \ell} = y^\ell$, $y^{3N_1 i + 2N_1 + \ell'} = -y^{N_1 + \ell'} - y^{\ell'}$ for every $i \geq 0$, $0 \leq \ell < 2N_1$, and $0 \leq \ell' < N_1$. Therefore, multiplication of any element of B_{N_1} by a power of y can be performed by at most one addition of two polynomials in B_{N_1} and sign inversion of it, that is, in at most $4N_1$ additive operations in k (again, if we count a sign inversion as an operation with unit cost, otherwise it is just $2N_1$). Overall, this step requires $O(N_1 \cdot N_2 \log N_2) = O(N \log N)$ operations in k.

Multiply component-wise two vectors of length $3N_2$, $\tilde{\bar{a}}$ and $\tilde{\bar{b}}$. Note, however, that only $2N_2$ out of these products are enough, namely only $\tilde{\bar{a}}_i \cdot \tilde{\bar{b}}_i$ where $i \not\equiv 0 \pmod 3$. This is explained in the next step.

Compute inverse DFT of $(\tilde{\bar{c}}_0, \ldots, \tilde{\bar{c}}_{3N_2-1})$ in $O(N \log N)$ operations in k. This computes the coefficients of $\bar{c}'(\bar{x}) = \bar{a}(\bar{x})\bar{b}(\bar{x}) \pmod{\bar{x}^{3N_2} - 1}$, and we need

$$\bar{c}(\bar{x}) = \bar{a}(\bar{x})\bar{b}(\bar{x}) \pmod{x^{2N_2} + x^{N_2} + 1}.$$

This is resolved by noticing that

$$\bar{c}_i = \bar{c}'_i - \bar{c}'_{i+2N_2}, \qquad \bar{c}_{i+N_2} = \bar{c}'_{i+N_2} - \bar{c}'_{i+2N_2},$$

for all $i = 0, \ldots, N_2 - 1$. To compute these differences, consider the explicit formulas of the direct DFT of order $3N_2$ with respect to ψ:

$$\tilde{\bar{c}}_{3i+j} = \sum_{\nu=0}^{3N_2-1} \bar{c}'_\nu \psi^{3i\nu + j\nu} = \sum_{\nu=0}^{N_2-1} \bar{c}'_{\nu,\,j} \psi^{3i\nu} =: \tilde{\bar{c}}_{i,\,j},$$

$$\bar{c}'_{i,\,j} = \frac{1}{N_2} \sum_{\nu=0}^{N_2-1} \tilde{\bar{c}}_{\nu,\,j} \psi^{-3i\nu}, \tag{6}$$

for $0 \le i < N_2$, $0 \le j \le 2$ and $\bar{c}'_{i,\,j} = \frac{1}{3}(\bar{c}'_i + \psi^{jN_2}\bar{c}'_{i+N_2} + \psi^{2jN_2}\bar{c}'_{i+2N_2}) \cdot \psi^{ij}$. Therefore,

$$\bar{c}'_{i+jN_2} = \frac{1}{3}(\bar{c}'_{i,\,0} + \psi^{-2jN_2-i}\bar{c}'_{i,\,1} + \psi^{-jN_2-2i}\bar{c}'_{i,\,2})$$

and the required differences

$$\bar{c}'_i - \bar{c}'_{i+2N_2} = \frac{1}{3}\big((\psi^{-i} - \psi^{-N_2-i})\bar{c}'_{i,\,1} + (\psi^{-2i} - \psi^{-2N_2-2i})\bar{c}'_{i,\,2}\big),$$
$$\bar{c}'_{i+N_2} - \bar{c}'_{i+2N_2} = \frac{1}{3}\big((\psi^{-2N_2-i} - \psi^{-N_2-i})\bar{c}'_{i,\,1} + (\psi^{-2i} - \psi^{-N_2-2i})\bar{c}'_{i,\,2}\big),$$

can be computed from $\bar{c}'_{i,\,j}$ for $j = 1, 2$, which can be computed via (6) from $\tilde{\bar{c}}_{i,\,j} = \tilde{\bar{c}}_{3i+j} = \tilde{\bar{a}}_{3i+j}\tilde{\bar{b}}_{3i+j}$ for $i = 0, \ldots, N_2 - 1$ and $j = 1, 2$, that is from $2N_2$ products.

Unembed in the similar way as in the original Schönhage-Strassen's algorithm. This requires $O(N)$ operations in k.

If we denote again $L'_{\mathcal{B}}(N)$ the total complexity of multiplication in B_N via Schönhage's algorithm $\mathcal{B}$, we obtain following complexity inequality:

$$L_{\mathcal{B}}(n) \le L'_{\mathcal{B}}(N) \le 2N_2 L'_{\mathcal{B}}(N_1) + O(N \log N).$$

It implies $L'_{\mathcal{B}}(N) = O(N \log N \log\log N)$ and the desired estimates (4) since $N \le 3n - 2$. Again, a more careful examination of the numbers of additions and multiplications used again gives also the upper bounds (4).

3.2.3 Cantor-Kaltofen's Generalization

In [11] Cantor and Kaltofen presented a generalized version of Schönhage-Strassen's algorithm [24], an algorithm $\mathcal{C}$ which computes the coefficients of a product of two polynomials over an arbitrary, not necessarily commutative, not necessarily associative algebra with unity with upper bounds (4). Here we present a simplified version of this algorithm which works over fields, or, more generally, over division algebras. We will use this restriction to perform divisions by constants of an algebra via multiplication by inverses of these constants.

Let $\omega \in \mathcal{C}$ be a primitive n-th root of unity. Then $\Phi_n(x) = \prod_{(i,\,n)=1}(x - \omega^i)$ is called a *cyclotomic polynomial* of order n. One easily deduces that for each n,

$$\Phi_n(x) \mid (x^n - 1) = \prod_{0 \le i < n} (x - \omega^i).$$

It is well known, that all coefficients of $\Phi_n(x)$ are integers, for every n, $\Phi_n(x)$ is irreducible over $\mathbb{Q}$, and or any s, n, $\Phi_{s^n}(x) = \Phi_s(x^{n-1})$. The degree of $\Phi_n(x)$ is

the number of natural numbers $i \le n$, coprime with n, which is denoted by $\phi(n)$ and called *Euler's totient function*. Trivially, $\phi(n) \le n-1$ with an equality iff n is a prime, and for $n \ge 3$, $\phi(n) > \frac{1}{2} \cdot \frac{n}{\log n}$, see [20]. From the above properties of $\Phi_n(x)$ we also have $\phi(s^n) = s^{n-1}\phi(s)$ for all s, $n \ge 1$. Therefore, if s is a constant and n grows, then the number of monomials in $\Phi_{s^n}(x)$ is bounded by a constant (for example, $s-1$).

Let k be a field of characteristic p, and $s \ge 2$ be some integer, that will be fixed throughout of the entire algorithm, $p \nmid s$. Cantor-Kaltofen's algorithm takes two degree $n-1$ polynomials $a(x)$ and $b(x)$ over k for $n \ge s^3$ and proceeds as follows:

Embed and pad Set $\nu \coloneqq \lceil \log_s\big((4n-2)\log s\big)\rceil$, such that

$$N \coloneqq \phi(s^\nu) \ge 2n-1.$$

The multiplication is then performed in $C_N \coloneqq k[x]/\Phi_N(x)$, where x is a principal N-th root of unity.

Extend Set $N_1 \coloneqq s^{\lfloor \frac{\nu}{2} \rfloor}\phi(s)$, $N_2 \coloneqq s^{\lceil \frac{\nu}{2} \rceil - 1}$ such that for

$$N_3 \coloneqq s^{\lfloor \frac{\nu}{2} \rfloor + 1}, \qquad N_1 = \phi(N_3) \ge N_2, \qquad N_1 N_2 = N.$$

Note, that $sN_2 \mid N_3$. Encode polynomials $a(x)$ and $b(x)$ (considered as elements of C_N) as polynomials of degree $N_2 - 1$ over C_{N_3}:

$$a(x) = \sum_{i=0}^{N-1} a_i x^i \mapsto \sum_{i=0}^{N_2-1} \underbrace{\left(\sum_{j=0}^{N_1-1} a_{i+N_2 j} y^j \right)}_{=:\bar{a}_i \in C_{N_3}} \bar{x}^i =: \bar{a}(\bar{x}).$$

y is a principal N_3-th root of unity in C_{N_3}, therefore, $\psi = y^{\frac{N_3}{N_2}}$ is a principal N_2-th root of unity and $\xi = y^{\frac{N_3}{sN_2}}$ is a principal sN_2-th root of unity in C_{N_3}.

Note, that $x^{N_1} \mapsto y$, and the polynomials $\bar{a}_i = \bar{a}_i(y)$ are in fact of degree at most $\lceil \frac{N_1-1}{2} \rceil$. This follows from the fact, that $a_l = 0$ for $l \ge n$, that is, for $i + N_2 j \ge n$, for $0 \le i < N_2$ and $0 \le j < N_1$. One can easily verify, that it is equivalent to the inequality $j \le \frac{n}{N_2} - 1 \le \frac{N-1}{2N_2} - 1 \le \lfloor \frac{N_1}{2} \rfloor - 1$. Therefore, multiplication of any two polynomials taken from the linear span of $\bar{a}_i$ modulo $\Phi_{N_3}(y)$ is in fact the ordinary multiplication of these polynomials.

Compute DFTs $\tilde{\bar{a}} = \mathrm{DFT}^{\psi}_{N_2}(\bar{a}(\bar{x}))$, $\tilde{\bar{a}}' = \mathrm{DFT}^{\psi}_{N_2}(\bar{a}(\xi\bar{x}))$, $\tilde{\bar{b}} = \mathrm{DFT}^{\psi}_{N_2}(\bar{b}(\bar{x}))$, and $\tilde{\bar{b}}' = \mathrm{DFT}^{\psi}_{N_2}(b(\xi\bar{x}))$. Precomputation of coefficient of $a(\xi x)$ and $b(\xi x)$ requires $O(N_2)$ multiplications by small powers of y in C_{N_3}. Computation of the DFTs requires $O(N_2 \log N_2)$ additions and multiplications by powers of ψ, that is, by powers of y, in C_{N_3}. Note, that, as usual, addition of two elements in C_{N_3} requires $N_2 = \phi(N_3)$ additions of elements in k.

Multiplications by powers of ψ, that is, by powers of y, can be first performed modulo $X^{N_3}-1$ at no cost (since they are in this case simply cyclic shifts), and then by reduction modulo $\Phi_{N_3}(x)$. This is possible since $\Phi_{N_3}(x)$ divides $x^{N_3}-1$. Since $\Phi_{N_3}(x)$ is monic and has at most s nonzero monomials, such a reduction can be performed with at most $(s-1)(N_3-N_2)=O(N_3)$ scalar multiplications and the same number of additions of elements in k. Therefore, the total cost of this step is $O(N_3 \cdot N_1 \log N_1) = O(N \log N)$ since $N_3 \le \frac{s}{\phi(s)} N_2 \le 2 \log s \cdot N_2 = O(N_2)$ and $N_1 N_2 = N$.

Multiply component-wise two pairs of vectors of length N_2: $\tilde{\bar{c}}'' = \tilde{\bar{a}} \cdot \tilde{\bar{b}}$ and $\tilde{\bar{c}}' = \tilde{\bar{a}}' \cdot \tilde{\bar{b}}'$. This is performed recursively by the same procedure since the components of these vectors are elements in C_{N_3}.

Compute inverse DFTs $\bar{c}' = \mathrm{DFT}_{N_2}^{\psi^{-1}}(\tilde{\bar{c}}')$ and $\bar{c}'' = \mathrm{DFT}_{N_2}^{\psi^{-1}}(\tilde{\bar{c}}'')$. This requires again $O(N \log N)$ steps, as in the computation of the direct DFTs.

Now recall, that we need the coefficients $\bar{c}_i \in C_{N_3}$ of the product of polynomials $\bar{c}(x) = \bar{a}(x)\bar{b}(x) \pmod{\Phi_N(x)}$. For this, we shall first compute the coefficients $\hat{c}_0, \ldots, \hat{c}_{2N_2-2}$ of the regular polynomial product $\hat{c}(x) = \bar{a}(x)\bar{b}(x)$. These can easily be computed from the $\bar{c}_i'$, $\bar{c}_i''$ via the following formulas for $0 \le i < N_2$:

$$\hat{c}_i = \frac{1}{N_2(1-\xi^{N_2})}(\bar{c}_i'' - \xi^{N_2}\bar{c}_i'), \qquad \hat{c}_{N_2+i} = \frac{1}{N_2(1-\xi^{N_2})}(\bar{c}_i' - \bar{c}_i'').$$

In order to get rid of divisions in C_{N_3} we can use the identity

$$\frac{1}{1-\xi^{N_2}} = \frac{1}{\tau} \prod_{\substack{2 \le i < s, \\ (i,\, s)=1}} (1-\xi^{N_2 i}),$$

where $\tau = 1$ if s is not a prime power, and $\tau = p$ if $s = p^\kappa$ for some prime p. Note, that in the latter case necessarily $\mathrm{char}\, k \neq p$. This identity shows how one can compute the fraction $\frac{1}{1-\xi}$ in $2\phi(s)-1$ additions and multiplications by powers of y in C_{N_3} without divisions: multiplication of the intermediate product Π by the next factor $1-\xi^{N_2 i}$ can be computed as $\Pi - \xi^{N_2 i}\Pi$. Therefore, all coefficients $\hat{c}_i$ for $0 \le i \le 2N_2-2$ can be computed in $O(N)$ operations in k. In order to obtain the coefficients of $\bar{c}(x)$, it suffices to reduce the polynomial $\hat{c}(x)$ modulo $\Phi_{N_3}(x)$ which can be performed in $O(N)$ steps, as explained before.

Unembedding in this case is not needed because of the choice of the encoding of polynomials: coefficients $\bar{c}_i$ computed in the Multiplication step, decoded back by substituting $y \mapsto x^{N_2}$, turn into polynomials in x with monomials of pairwise different degrees for different $i = 0, \ldots, N-1$.

If we denote $L'_{\mathcal{C}}(N)$ the total complexity of multiplication in C_N via Cantor-Kaltofen's algorithm $\mathcal{C}$, we obtain following complexity inequality:

$$L_{\mathcal{C}}(n) \le L'_{\mathcal{C}}(N) \le 2N_2 L'_{\mathcal{C}}(N_1) + O(N \log N).$$

The choice of parameters N_1 and N_2 implies $L'_{\mathcal{C}}(N) = O(N \log N \log\log N)$ and the desired estimates (4) since $N < (s-1)\log s \cdot (2n-1) = O(n)$. A more careful examination of the numbers of additions and multiplications used again gives also the upper bounds (4).

If $\operatorname{char} k \neq 2$ and $s = 2$, then $\Phi_{s^\nu}(x) = x^{2^{\nu-1}} + 1$, and we get the multiplication in the algebra $A_{2^{\nu-1}}$ from the Schönhage-Strassen's algorithm. If $\operatorname{char} k \neq 3$ and $s = 3$, then $\Phi_{s^\nu}(x) = x^{2\cdot 3^{\nu-1}} + x^{3^{\nu-1}} + 1$ and we get the multiplication in the algebra $B_{3^{\nu-1}}$. However, the multiplication is performed differently: instead of performing one DFT of order $N_2 \sim 2\sqrt{N}$ over A_{N_1} (of order $3N_2$ over B_{N_1} with only $2N_2 \sim 2\sqrt{N}$ multiplications sufficient, resp.), Cantor-Kaltofen's algorithm performs two DFTs of order $N_2 \sim \sqrt{N}$ over C_{N_3}.

Summarizing the above algorithms of complexity $O(n \log n \log\log n)$ we notice that in case, when it is impossible to apply FFT directly in the ground field, a ring extension is always introduced. Since the costs of all recursive steps are roughly the same, total complexity of such an algorithm can be naturally bounded by the product of the cost of one recursive step by the number of steps, which is $O(\log\log n)$ in the algorithm $\mathcal{B}$. Complexity of one recursive step is defined by the complexity of computing DFTs, for which nothing better than $O(n \log n)$-time algorithms for computing of a DFT of order n is currently known. The first potential improvement of this scheme is to reduce the complexity of algorithms computing DFT. The second is reducing the number of recursive steps of such an algorithm. In the first case we can increase the number of recursive steps needed, depending on the boost we will achieve in computing DFT. In the second case we can increase the number of operations used by DFT computations, however, we must always make sure that the product of these two values does not exceed $\Omega(n \log n \log\log n)$. In this paper we are concerned mostly with the problem of reduction of the recursive depth of such algorithms. Effectivity of our solution appears to depend only on algebraic properties of the ground field.

4 An Upper Bound for the Complexity of DFT

In this section we summarize the best known upper bounds for the computation of DFTs over an algebra A with unity 1. Let $\omega \in A$ be a principal n-th root of unity. For $a(x) \in A[x]$ of degree $n-1$ let $\tilde{a} = \mathrm{DFT}_n^\omega(a(x)) \in A^n$. We will denote the total number of operations over A that are sufficient for an algebraic algorithm to compute the DFT of order n over A by $D_A(n)$. In case, when the algebra A be insignificant or clear from the context, we will use the notation $D(n)$.

There is always an obvious way to compute $\tilde{a}$ from the coefficients of $a(x)$.

Lemma 1. *For every A and $n \geq 1$, such that the DFT of order n is defined over A,*

$$D_A(n) \leq \begin{cases} 2n^2 - 3n + 1, & \text{if } 2 \nmid n, \\ 2n^2 - 5n + 4, & \text{if } 2 \mid n. \end{cases} \tag{7}$$

Proof. To compute $\tilde{a}_0$, $n-1$ additions are always sufficient. Let $\omega \in A$ be a principal n-th root of unity. If $2 \mid n$, then $\omega^{\frac{n}{2}} = -1$, and to compute $\tilde{a}_{\frac{n}{2}}$, $n-1$ additions/subtractions are also sufficient. For the rest of the coefficients $\tilde{a}_i$, one always needs $n-1$ additions and, in case of odd n, $n-1$ multiplications by powers of ω. For even n, one multiplication can be saved, namely, by $\omega^{i\frac{n}{2}} = (-1)^i$, it can be implemented by selective changing the sign of the corresponding additive operation in the sum for $\tilde{a}_i$. Therefore, we obtain

$$D_A(n) \le \begin{cases} (n-1) + 2(n-1)^2 = 2n^2 - 3n + 1, & \text{if } 2 \nmid n, \\ 2(n-1) + (n-2)((n-2) + (n-1)) = 2n^2 - 5n + 4, & \text{if } 2 \mid n, \end{cases}$$

which proves the statement. □

The next method of effective reduction of a DFT of large order to DFTs of smaller orders is known as Cooley-Tukey's algorithm [14], [13, Section 4.1] and is based on the following lemma which directly follows from the well-known facts and is present here for completeness.

Lemma 2. *Let the DFT of order*

$$n = p_1^{d_1} \dots p_s^{d_s} \ge 2 \tag{8}$$

be defined over A (p_σ are not necessary prime and even pairwise coprime). Then

$$D(n) \le n \sum_{\sigma=1}^{s} \left(\frac{d_\sigma}{p_\sigma} (D(p_\sigma) - 1) + d_\sigma \right) - n + 1. \tag{9}$$

Proof. We first prove that if $n = n_1 n_2$, then

$$D(n) \le n_1 D(n_2) + n_2 D(n_1) + (n_1 - 1)(n_2 - 1). \tag{10}$$

Let $\omega \in A$ be a principal n-th root of unity. Then $\omega_1 \coloneqq \omega^{n_2}$ is a principal n_1-th root of unity and $\omega_2 \coloneqq \omega^{n_1}$ is a principal n_2-th root of unity. For a polynomial $a(x) \in A[x]/(x^n - 1)$, consider $\tilde{a} = \mathrm{DFT}_n^\omega(a(x))$: for $0 \le j < n_2$, $0 \le l < n_1$

$$\tilde{a}_{n_1 j + l} = \sum_{\nu=0}^{n-1} a_\nu \omega^{(n_1 j + l)\nu} = \sum_{\nu=0}^{n_1-1} \sum_{\mu=0}^{n_2-1} a_{n_2\nu+\mu} \omega^{n_2\nu(n_1 j + l) + \mu(n_1 j + l)}$$

$$= \sum_{\mu=0}^{n_2-1} \Bigg(\omega^{\mu l} \underbrace{\sum_{\nu=0}^{n_1-1} a_{n_2\nu+\mu} \omega_1^{\nu l}}_{=: \tilde{a}_{\mu,\, l}} \Bigg) \omega_2^{\mu j} = \sum_{\mu=0}^{n_2-1} \underbrace{(\omega^{\mu l} \tilde{a}_{\mu,\, l})}_{=: \hat{a}_{\mu,\, l}} \omega_2^{\mu j} = \sum_{\mu=0}^{n_2-1} \hat{a}_{\mu,\, l} \omega_2^{\mu j} =: \tilde{a}_{j,\, l}.$$

Computation of all values $\tilde{a}_{j,\, l}$ for a fixed l can be performed via the DFT of order n_2 with respect to ω_2. Therefore, to compute all values $\tilde{a}_{j,\, l}$, i.e., all values a_i for $0 \le i < n$, it suffices to perform n_1 DFTs of order n_2. Computation of all values $\tilde{a}_{\mu,\, l}$ for fixed μ can be performed via the DFT of order n_1 with respect to ω_1. Therefore, to compute all values $\tilde{a}_{\mu,\, l}$, it suffices to perform n_2 DFTs of

order n_1. Finally, to compute $\hat{a}_{\mu,l}$ from $\tilde{a}_{\mu,l}$, one needs one multiplication by $\omega^{\mu l}$ if $\mu > 0$ and $l > 0$ (if $\mu = 0$ or $l = 0$ then no computation is needed). This takes $(n_1 - 1)(n_2 - 1)$ multiplications by powers of ω to compute all values $\hat{a}_{\mu,l}$. This proves (10).

(9) follows by consecutive application of (10) choosing d_1 times p_1 for n_1, then d_2 times p_2, etc. Noting that $D(1) = 0$ completes the proof. □

Corollary 1. *Let n be as in (8), and let all $2 = p_1 < p_2 < \cdots < p_s$ be all primes. Then*

$$D(n) \le \left(\frac{3}{2}d_1 + 2\sum_{\sigma=2}^{s} d_\sigma(p_\sigma - 1) - 1\right)n + 1. \tag{11}$$

In particular,

$$D(n) \le 2 \max_{1\le\sigma\le s} p_\sigma \cdot n \log n. \tag{12}$$

Proof. (11) follows from (9) by applying the upper bound of Lemma 1 for the values of $D(p_\sigma)$.

Obviously $d_1, \ldots, d_s \le \log n$ since $p_\sigma^{d_\sigma} \le n$, $p_\sigma \ge 2$ for $1 \le \sigma \le s$. Therefore,

$$D(n) \le \left(\frac{3}{2} + 2\big(\max_{1\le\sigma\le s} p_\sigma - 1\big) - 1\right) n \log n + 1 \le 2 \max_{1\le\sigma\le s} p_\sigma \cdot n \log n,$$

which proves (12). □

Lemma 2 provides an efficient method of reduction of a DFT of composite order n to several DFTs of smaller orders which divide n. For example, if all p_σ in (8) are bounded by some constant, then (12) shows that Cooley-Tukey's algorithm computes the DFT of order n in $O(n \log n)$ steps. Furthermore, if $\max_{1\le\sigma\le s} p_\sigma \le g(n)$ for some slowly growing function $g(n)$, say $g(n) = o(\log \log n)$, then (12) gives an upper bound of $o(n \log n \cdot g(n))$ for the computation of the DFT of order n. However, this method fails to be effective if n has large prime factors (or is just prime). We could use the algorithm from Lemma 1, but sometimes we can apply Rader's algorithm to compute a DFT of prime order [22], [13, Section 4.2].

Lemma 3. *Let p be a prime, and assume that the DFT of order p is defined over A.*

1. *If the DFT of order $p-1$ is defined over A, then $D(p) \le 2D(p-1) + O(p)$.*
2. *If for $n > 2p - 4$, the DFT of order n is defined over A, then*

$$D(p) \le 2D(n) + O(n).$$

Remark 1. Note, that the first bound can be efficient if $p-1$ is a smooth number. Otherwise we may choose some larger smooth n for the second case, making sure that the DFT of order n exists over A and n is not too large, that is, to achieve an $O(p \log p)$ upper bound for $D(p)$.

Proof. Let $\omega \in A$ be a principal p-th root of unity. For a polynomial

$$a(x) \in A[x]/(x^p - 1),$$

the value of $\tilde{a}_0 = \sum_{i=0}^{p-1} a_i$ can be computed directly by performing $p-1$ additions. For $1 \le i \le p-1$,

$$\tilde{a}_i - a_0 = \sum_{j=1}^{p-1} a_j \omega^{ij} =: \tilde{a}'_i. \tag{13}$$

Thus, to compute all $\tilde{a}_i$ from $\tilde{a}'_i$, $p-1$ additions are enough.

1. The multiplicative group $\mathbb{F}_p^* = \{1 \le i < p\}$ is isomorphic to the cyclic group $\mathbb{Z}_{p-1}$ with $p-1$ elements. We will denote the isomorphism by α. For $a''_{i-1} \coloneqq a_{\alpha(i)}$ and $\tilde{a}''_{i-1} = \tilde{a}'_{\alpha(i)}$, from (13) we obtain

$$\tilde{a}''_i = \sum_{j=1}^{p-1} a_j \omega^{\alpha(i)+\alpha(j)} = \sum_{j=0}^{p-2} a''_j \omega^{\alpha(i+j)}.$$

   The latter is a cyclic convolution, which can be performed via computing the coefficients of the product of the degree $p-2$ polynomial

$$a''(x) = \sum_{i=0}^{p-2} a''_i x^i,$$

   and the degree $p-2$ polynomial with fixed coefficients

$$\omega(x) = \sum_{i=0}^{p-2} \omega^{\alpha(i)} x^i.$$

   This can be achieved by computing the DFT of $a''(x)$, performing $p-1$ multiplications by constants (components of the DFT of $\omega(x)$, in fact, these are just polynomials in ω), and computing the reverse DFT. This proves the first bound.

2. For an $n \ge 2p-3$, we may define the polynomials

$$\hat{a}(x) = a''_0 + a''_1 x^{n-p+2} + \cdots + a''_{p-2} x^{n-1}, \quad \hat{\omega}(x) = \sum_{i=0}^{n-1} \omega^{\alpha(i \bmod (p-2)+1)} x^i$$

   and compute their cyclic convolution. Then the first $p-1$ coefficients of the cyclic convolution will be exactly the $a''_0, \ldots, a''_{p-2}$. Note, that again, we do not need to count the complexity of the DFT of $\hat{\omega}(x)$ since it is fixed and can be precomputed. This proves the second bound. □

Corollary 2. *Let p be a fixed odd prime, k be a field where the DFT of order $p^N - 1$ is defined for $N = 2^n$, $n \ge \lceil \log(2p-5) \rceil$. Then $D_k(p^N - 1) = O(p^N \cdot N^2)$.*

Proof. We have $p^N - 1 = (p-1)(p+1)(p^2+1)\cdots(p^{2^{n-1}}+1)$. Since p is odd, each factor is even and $p^N - 1 = 2^n \cdot \frac{p-1}{2} \prod_{i=1}^{n-1} \frac{p^{2^i}+1}{2}$. Let $p^N - 1 = p_1^{d_1} \cdots p_s^{d_s}$ be the decomposition of $p^N - 1$ into primes and $p_1 = 2 < p_2 < \cdots < p_s$, and $p_2, \ldots, p_{i_1}$ are all less than $\frac{p-1}{2}$, $p_{i_1+1}, \ldots, p_{i_2}$ are less than $\frac{p+1}{2}$, and, in general, $p_{i_j+1}, \ldots, p_{i_{j+1}}$ are less or equal than $\frac{p^{2^{j-1}}+1}{2}$. Note, that $i_n = s$. We also set $i_{-1} = 0$, $i_0 = 1$. From (9) we have

$$D(p^N - 1) \leq (p^N - 1) \sum_{\sigma=1}^{s} \left(\frac{d_\sigma}{p_\sigma} (D(p_\sigma) - 1) + d_\sigma \right) - p^N + 2.$$

Obviously, for $p_1 = 2$, we have $D(p_1) = 2 \leq p_1 \cdot \log p_1$. Using Lemma 3 we can compute the DFT of orders p_σ for $p_\sigma = 2, \ldots, i_2$ in $8p_\sigma \log p_\sigma + O(p_\sigma)$ time since we can reduce each DFT of order p_σ to 2 DFTs of order $2^{n_1} > 2p_\sigma - 4$, $2^{n_1} < 4p_\sigma$. This is possible since the DFT of order $2^n > 2 \cdot \frac{p-1}{2} - 4$ is defined over k. In the same way, the DFT of order p_σ for $\sigma = i_1 + 1, \ldots, i_2$ can be computed in $16p_\sigma \log p_\sigma + O(p_\sigma)$ steps since $2^n \cdot \frac{p-1}{2} > 2 \cdot \frac{p+1}{2} - 4$. Continuing this process we obtain the following upper bound:

$$D(p^N - 1) \leq (p^N - 1) \sum_{j=-1}^{n-1} \sum_{\sigma=i_j+1}^{i_{j+1}} O\left(d_\sigma \cdot 2^j \log p_\sigma + d_\sigma\right)$$

$$= O(p^N \cdot N \cdot \log \prod_{\sigma=1}^{s} p_\sigma^{d_\sigma}) = O(p^N \cdot N^2),$$

which completes the proof. □

Remark 2. For a fixed odd prime p, the DFT of order $p^{2^n}-1$ is defined in the field $\mathbb{F}_{p^{2^n}}$ since the multiplicative group $\mathbb{F}^*_{p^{2^n}}$ of order $p^{2^n} - 1$ is cyclic. Corollary 2 implies that the DFT of order $p^{2^n}-1$ can be computed in $O(p^{2^n} \cdot 2^{2n})$ steps over $\mathbb{F}_p$. A similar argument shows that the same holds for any field of characteristic p which contains $\mathbb{F}_{p^{2^k}}$ as a subfield.

5 Unified Approach for Fast Polynomial Multiplication

In this section we present our main contribution. We proceed as follows: first we introduce the notions of the degree function and of the order sequence of a field. Then we describe the DFT-based algorithm $\mathcal{D}_k$ which computes the product of two polynomials over a field k. We show that $\mathcal{D}_k$ generalizes any algorithm for polynomial multiplication that relies on consecutive applications of DFT, and in particular, Schönhage-Strassen's [24], Schönhage's [23], and Cantor-Kaltofen's [11] algorithms for polynomial multiplication are special cases of the algorithm $\mathcal{D}_k$. We prove that both the upper and the lower bounds for the total complexity of the algorithm $\mathcal{D}_k$ depend on the degree function of k

and the existence of special order sequences for k. In particular, we show that $L_{\mathcal{D}_k}(n) = \Omega(n \log n)$ when k is a finite field, and $L_{\mathcal{D}_{\mathbb{Q}}}(n) = \Omega(n \log n \log\log n)$. Furthermore, we show sufficient conditions on the field k for the algorithm $\mathcal{D}_k$ to compute the product of two degree n polynomials in $o(n \log n \log\log n)$, that is, to outperform Schönhage-Strassen's, Schönhage's and Cantor-Kaltofen's algorithms. Finally, we pose a number-theoretic conjecture whose validity would imply faster polynomial multiplication over arbitrary fields of positive characteristic.

In what follows k always stands a field.

5.1 Extension Degree and Order Sequence

Definition 1. The *degree function* of k, is $f_k(n) = [k(\omega_n) : k]$ for any positive n, where ω_n is a primitive n-th root of unity in the algebraic closure of k.

For example, $f_k(n) = 1$ if k is algebraically closed, $f_{\mathbb{R}}(n) = 1$ if $n \leq 2$ and $f_{\mathbb{R}}(n) = 2$ for $n \geq 3$, $f_{\mathbb{Q}}(n) = \phi(n)$ where $\phi(N)$ is as before the Euler's totient function.

An important idea behind Fürer's algorithm [16, 15] is a field extension of small degree containing a principal root of unity of high smooth order. In case of integer multiplication, the characteristic of the ground ring is a parameter we can choose [15], and it allows us to pick such $\mathbb{Z}_{p^c}$ that $p^c - 1$ has a large smooth factor. However, in case of multiplication of polynomials over fields, we cannot change the characteristic of the ground field. In what follows we explore this limitation.

Definition 2. An integer $n > 0$ is called *c-suitable* over the field k, if the DFT of order n is defined over k and $D_k(n) \leq cn \log n$.

It follows from Corollary 1 that any c-smooth n is c-suitable over k as long as the DFT of order n is defined over k, and Lemma 3 also implies, that if for each prime divisor p of n, p, or $p-1$ or some $n' \geq 2p-3$, $n' = O(p)$ is c-suitable over k, then n is $O(c)$-suitable. If $\operatorname{char} k \geq 3$, then the integers $(\operatorname{char} k)^{2^n} - 1$ are 2^n-suitable over k for arbitrary n (see Remark 2).

Definition 3. Let $s(n) : \mathbb{N} \to \mathbb{R}$ be such that $s(n) > 1$. A sequence

$$\mathcal{N} = \{n_1, n_2, \dots\}$$

is called an *order sequence* of sparseness $s(n)$ for the field k, if

$$n_i < n_{i+1} \leq s(n_i) n_i$$

and $n_i \mid n_{i+1}$ for $i \geq 1$, and $n_i = n'_i n''_i$, such that there exists a ring extension of k of degree n'_i containing an n''_i-th principal root of unity $\omega_{n''_i}$, which is $O(1)$-suitable over this extension. If $s(n) \leq C$ for some constant C, then $\mathcal{N}$ is called an order sequence of *constant sparseness*.

It follows from Remark 2 that $n_i = 2^i \cdot (p^{2^i} - 1)$ is almost an order sequence of sparseness $s(n) = n$ for any field of characteristic p. Decreasing the upper bound for the computation of DFT from $O(n \log^2 n)$ to $O(n \log n)$ would turn it into an order sequence.

Remark 3. If $\operatorname{char} k \neq 2$, then for the order sequence $\mathcal{N} = \{2^i\}_{i \geq 1}$, $f_k(n'') \leq \frac{n''}{2}$ for each $n = n'n'' \in \mathcal{N}$ since if for $n \in \mathcal{N}$, $\omega_{n''}$, $n'' = 2^{\lceil \frac{i-1}{2} \rceil}$, $n'' = 2^{\lfloor \frac{i-1}{2} \rfloor + 1}$ is a primitive n''-th root of unity in the algebraic closure of k, then

$$k(\omega_{n''}) \cong k[x]/p(x)$$

and $p(x) \mid x^{\frac{n''}{2}} + 1$. The same argument shows that if $\operatorname{char} k \neq 3$ and

$$\mathcal{N} = \{2 \cdot 3^i\}_{i \geq 1},$$

then $f_k(n'') \leq \frac{2n''}{3}$ for each $n = n'n'' \in \mathcal{N}$, $n' = 2 \cdot 3^{\lceil \frac{i-1}{2} \rceil}$, $n'' = 3^{\lfloor \frac{i-1}{2} \rfloor + 1}$, since for $k(\omega_{n''}) \cong k[x]/p(x)$, $p(x) \mid x^{\frac{2n''}{3}} + x^{\frac{n''}{3}} + 1$. Both these order sequences have constant sparsenesses.

Definition 4. A field k is called

- *Fast*, if there is an order sequence $\mathcal{N}$ of constant sparseness such that $f_k(n'_i) = O(1)$ for all $n_i = n'_i n''_i \in \mathcal{N}$;
- $t(n)$-*Fast*, if there exists an order sequence $\mathcal{N}$ of constant sparseness such that $f_k(n'_i) \leq t(n'_i)$ for all $n_= n'_i n''_i \in \mathcal{N}$.
- $t(n)$-*Slow*, if for any order sequence $\mathcal{N}$ of constant sparseness,

  $$f_k(n'_i) \geq t(n'_i)$$

  for all $n_i = n'_i n''_i \in \mathcal{N}$.

For example, any algebraically closed field is fast, $\mathbb{R}$ is a fast field, and $\mathbb{Q}$ is a $\phi(n)$-slow field, in particular, $\mathbb{Q}$ is an $\frac{n}{2 \log n}$-slow field. It follows from Remark 3, that any field of characteristic different from 2 is $\frac{n}{2}$-fast, and any field of characteristic different from 3 is $\frac{2n}{3}$-fast.

If we want to extend a $b(n)$-slow field k with an n-th root of unity, the degree of the extension will be $\Omega(b(n))$. We will see, that to increase performance of a DFT-based algorithm for computing the product of two degree $n-1$ polynomials over k, we need to take an extension $K \supseteq k$ of degree n_1 over k, such that K contains a primitive n_2-th root of unity. We will want n_2 to be a large suitable number and to belong to a "not too sparse" order sequence, preferably of constant sparseness, n_1 to be small such that $2n - 1 \leq n_1 n_2 = O(n)$.

We close this subsection with introducing some technical notation. for a function $f : \mathbb{N} \to \mathbb{N}$, such that $\limsup_{n \to \infty} f(n) = \infty$, we will denote by $f^{\vee}(n)$ the minimal value $f(i)$ over all integer solutions i of the inequality

$$i \cdot f(i) \geq n.$$

For example, $n^{\vee} = \lceil \sqrt{n} \rceil$, $\left(\frac{n}{\log n}\right)^{\vee} \sim \sqrt{\frac{n}{\log n}}$ for $n \geq 2$,[2] and for $q \geq 2$,

[2] By $f(n) \sim g(n)$ we denote $f(n) = (1 \pm o(1))g(n)$.

$(\log_q n)^\vee = \log_q n - \Theta(\log_q \log_q n)$ if $n \geq q$.

We will need to restrict the possible values for i in the inequality to be taken from some order sequence.

For a monotonically growing function $f : \mathbb{N} \to \mathbb{N}$, such that $\lim_{n\to\infty} \frac{f(n)}{n} < 1$, we will define $f^{(0)}(n) = n$, and for $i \geq 1$, $f^{(i)}(n) = f^{(i-1)}(f(n))$. For each $n \geq 1$, there exists the value $i = i(n)$ such that

$$f^{(i-1)}(n) \neq f^{(i)}(n) = f^{(i+1)}(n) = \cdots .$$

This value will be denoted by $f^*(n)$. For example,

$$\left(\left\lceil \frac{n}{2} \right\rceil\right)^* = \lceil \log n \rceil , \qquad \left(\left\lceil \sqrt{n} \right\rceil\right)^* = \lceil \log \log n \rceil , \qquad (\lceil \log n \rceil)^* = \lceil \log^* n \rceil .$$

5.2 Generalized Algorithm For Polynomial Multiplication

The DFT-based algorithm $\mathcal{A}$, the Schönhage-Strassen's and Schönhage's algorithms $\mathcal{B}$, and the Cantor-Kaltofen's algorithm $\mathcal{C}$ are all based on the idea of a field extension with roots of unity of large smooth orders to reduce the polynomial multiplication to many polynomial multiplications of smaller degrees by means of DFT. The natural metaflow of all these algorithms can be generalized as follows: let $\mathcal{N}$ be an order sequence of constant sparseness over a field k, for two polynomials $a(x)$ and $b(x)$ of degree $n-1$ over k:

Embed Choose a polynomial $P_N(x)$ of degree $N = N'N'' \in \mathcal{N}$,

$$2n - 1 \leq N = O(n),$$

and switch to multiplication in $A_N \coloneqq k[x]/P_N(x)$. From this moment consider $a(x)$ and $b(x)$ as elements of A_N. There should be an efficiently computable by means of DFTs injective homomorphism $\psi : A_N \to (A_{N'})^{2N''}$, where $A_{N'} \cong k[y]/P_{N'}(y)$ for some $P_{N'}(y) \in k[y]$, and $A_{N'}$ contains a principal N''-th (or $2N''$-th) root of unity.

Transform By means of DFTs over $A_{N'}$ compute

$$\tilde{a} \coloneqq \psi(a(x)), \qquad\qquad \tilde{b} \coloneqq \psi(b(x)),$$

both in $(A_{N'})^{2N''}$.

Multiply Compute $2N''$ products $\tilde{c} \coloneqq \tilde{a} \cdot \tilde{b}$ in $A_{N'}$.

Back-Transform By means of DFT compute $c(x) = \psi^{-1}(\tilde{c})$, which is the ordinary product of the input polynomials.

Unembed Reduce the product modulo $P_N(x)$ to return the product in A_N.

Theorem 1. *The algorithm $\mathcal{A}$, Schönhage-Strassen's and Schönhage's algorithms $\mathcal{B}$ and Cantor-Kaltofen's algorithm $\mathcal{C}$ are instances of the algorithm $\mathcal{D}$.*

Proof. For a field k which contains an N-th primitive root of unity for

$$N = 2^{\lceil \log(2n-1) \rceil},$$

$N = O(n)$, set $P_N(x) = x^N - 1$, $N' = 1$, $N'' = N$ and $A_{N'} = k$. Then ψ is the DFT of order $2N$ (which can be trivially reduced to N in this case) over k and the algorithm $\mathcal{D}$ appears to be the algorithm $\mathcal{A}$.

For a field k of characteristic different from 2, for $\nu = \lceil \log(2n-1) \rceil$ and $N = 2^\nu$, set $P_N(x) = x^N + 1$, $N' = 2^{\lceil \frac{\nu}{2} \rceil}$, and $N'' = 2^{\lfloor \frac{\nu}{2} \rfloor}$. Then ψ is the DFT of order $2N''$ over $A_{N'}$ and the algorithm $\mathcal{D}$ appears to be the Schönhage-Strassen's algorithm $\mathcal{B}$ [24].

For $\operatorname{char} k = 2$, set $\nu = \lceil \log_3(n - \frac{1}{2}) \rceil$, $N = 3^\nu$, and $P_{2N}(x) = x^{2N} + x^N + 1$, $N' = 3^{\lceil \frac{\nu}{2} \rceil}$, and $N'' = 3^{\lfloor \frac{\nu}{2} \rfloor}$. Then ψ is the DFT of order $3N''$ over $A_{N'}$. However, to fetch the entries of the product in $A_{N'}$ by means ψ^{-1}, $2N''$ products of polynomials in $A_{N'}$ are sufficient [23]. Therefore, the algorithm $\mathcal{D}$ appears to be the Schönhage's algorithm $\mathcal{B}$.

For an arbitrary field k fix a positive integer $s \neq \operatorname{char} p$ and find the least ν such that $N = \phi(s^\nu) = s^{\nu-1}\phi(s) \geq 2n-1$, and let $\hat{N} = s^\nu$. Set $P_{\hat{N}}(x) = \Phi_{\hat{N}}(x)$, $N' = \phi(s^{\lfloor \frac{\nu}{2} \rfloor + 1})$, and $N'' = s^{\lceil \frac{\nu}{2} \rceil - 1}$. Then $\psi = \alpha \circ \beta$ where α stands for 2 DFTs of order N'' over A', and β is a linear map $A_{N'}[x] \to A_{N'}[x] \times A_{N'}[x]$ such that $\beta(a(x)) = (a(x), a(\gamma x))$, where γ is the sN''-th root of unity in $A_{N'}$, i.e., for $A_{N'} \cong k[y]/\Phi_{\hat{N}'}(y)$, either $\gamma = y$ or $\gamma = y^2$. One can easily show that β and β^{-1} are computable in linear time. Therefore, the algorithm $\mathcal{D}$ appears to be the Cantor-Kaltofen's algorithm $\mathcal{C}$. □

5.3 Complexity Analysis

From the description of the algorithm $\mathcal{D}$ we have

$$L_{\mathcal{D}}(n) = L'_{\mathcal{D}}(N) = 2N''L'_{\mathcal{D}}(N') + 2T(\psi(N)) + T(\psi^{-1}(N))$$

where $L'_{\mathcal{D}}(N)$ denotes the complexity of $\mathcal{D}$ computing the product in A_N, $T(\psi(N))$ and $T(\psi^{-1}(N))$ stand for the total complexities of the transformations ψ and ψ^{-1} on inputs of length N respectively.

Theorem 2. *Let the algorithm $\mathcal{D}$ compute the product of two polynomials in A_N in ℓ recursive steps and let $N' = N'_\lambda$ and $N'' = N''_\lambda$ be chosen on the step $\lambda = 1, \ldots, \ell$ ($N'_0 = N$, $N'_\ell = O(1)$), and for $M(N'_\lambda) = \max\{1, \frac{M^*(N'_\lambda)}{N'_\lambda}\}$, where $M^*(N'_\lambda)$ stands for the complexity of multiplication of an element in $A_{N'_\lambda}$ by powers of an N''_λ-th root of unity (which exists in $A_{N'_\lambda}$ by assumption). Then*

$$L'_{\mathcal{D}}(N) = \Theta\left(N \cdot 2^\ell + N \sum_{\lambda=1}^{\ell} 2^{\lambda-1} \cdot M(N'_\lambda) \log N''_\lambda\right), \tag{14}$$

and if $\operatorname{char} k \neq 2$, *then*

$$L'_{\mathcal{D}}(N) = \Omega\left(N \cdot 2^{(f_k^\vee)^*(N)} + N \sum_{\lambda=1}^{(f_k^\vee)^*(N)-1} 2^{\lambda-1} \log(f_k^\vee)^{(\lambda)}(N)\right). \tag{15}$$

Proof. Consider the total cost of the algorithm with respect to the computational cost of the first step:

$$L'_{\mathcal{D}}(N) = 2N'' \cdot L'_{\mathcal{D}}(N') + \Theta\left(N'' \log N'' \cdot (N' + M^*(N'))\right). \tag{16}$$

This follows from the fact that we need to perform a DFT of order N'' over $A_{N'}$. Each DFT requires $\Theta(N'' \log N'')$ additions of elements in $A_{N'}$ and the same number of multiplications by powers of an N''-th principal root of unity. Since $\dim_k A_{N'} = N'$, one addition in $A_{N'}$ takes N' additions in k, and by definition, $M^*(N')$ is the number of operations in k, needed to computed the necessary products by powers of a principal root of unity. Unrolling (16) (by using (16) recursively ℓ times), (14) follows.

To obtain (15) from (14) we use the trivial lower bound $M(N') \geq 1$. We then notice that $N' \geq f_k^\vee(N'')$, therefore, we come to the equality $N''_\ell = O(1)$ not earlier than for $\ell = (f_k^\vee)^*(N)$, by definition of these operations and the lower bound (15) follows. □

Corollary 3.

1. *For an arbitrary fast field k, we have $L_{\mathcal{D}_k}(n) = O(n \log n)$.*
2. *For an $o(\log\log n)$-fast field k, we have $L_{\mathcal{D}_k} = o(n \log n \log\log n)$.*
3. *For an $\Omega(n^{1-o(1)})$-slow field k, we have $L_{\mathcal{D}_k} = \Omega(n \log n \log\log n)$.*

Proof.

1. By definition of a fast field, it suffices to take constant number of steps (in fact, even one step) to extend k with a principal root of unity of a suitable order. This means, $\ell = 1$ and $N' = O(1)$. Therefore, $M(N') = O(1)$ and trivially $\log N'' \leq \log N$.

2. By definition of an $o(\log\log n)$-fast field, in the first step we have

   $$N' = o(\log\log N).$$

   We always can bound $M(N'_i)$ with N'_i in (14), and we have

   $$\ell = o(\log^* \log^* n).$$

   Bounding the first summand in the sum in (14) by

   $$N \cdot N' \cdot \log N = o(n \log n \log\log n),$$

   and each next summand by $o(n \cdot 2^{\log^* \log^* n} \cdot \log\log n \cdot \log(\log\log n))$, we obtain the statement.

3. For $f_k(n) = \Omega(n^{1-o(1)})$ we have $f_k^\vee(n) = \Omega(n^{\frac{1}{2}-o(1)})$ and

   $$(f_k^\vee)^*(n) = \Omega(\log\log n).$$

   Each summand in (15) is therefore $\Omega(\log n)$ and the statement follows. □

Corollary 4. $L_{\mathcal{D}_{\mathbb{Q}}}(n) = \Omega(n \log n \log\log n)$.

Proof. We have $f_{\mathbb{Q}}(n) \geq \frac{n}{2\log n} = \Omega(n^{1-o(1)})$ and the statement follows from Corollary 3. □

Corollary 5. *For the finite field* $\mathbb{F}_p$, $L_{\mathcal{D}_{\mathbb{Q}}}(n) = \Omega(n \cdot \log n)$.

Proof. We have $f_{\mathbb{F}_p}(n) \sim \log_p n$ since the multiplicative group $\mathbb{F}_p^*$ is cyclic and in the extension field $\mathbb{F}_{p^n}$ of degree n exists a primitive root of unity of order $p^n - 1$. This means that $f^{\vee}_{\mathbb{F}_p}(n) \sim \log_p n$ and $(f^{\vee}_{\mathbb{F}_p})^*(n) \sim \log^*_p n$, and the statement follows from taking in (15) the first summand which is always $\Theta(n \log n)$. □

Note, that Theorem 2 does not give any pessimistic lower bound in case of finite fields. Actually, it can give a good upper bound if one can prove existence of order sequences of constant sparseness over finite fields. More formally,

Corollary 6. *Assume, there exists an order sequence* $\mathcal{N} = \{n_i(p^{n_i} - 1)\}_{i \geq 1}$ *of constant sparseness over* $\mathbb{F}_p$ *and assume that the complexity of multiplication by powers of a principal* $(p^{n_i} - 1)$*-th root of unity in* $\mathbb{F}_{p^{n_i}}$ *can be performed in* $O(n_i)$ *time. Then* $L_{\mathcal{D}_{\mathbb{F}_p}}(n) = O(n \log n \log^* n)$.

Proof. From (16) we get $L'_{\mathcal{D}_{\mathbb{F}_p}}(N) \leq \frac{2N}{\log_p N} L'_{\mathcal{D}_{\mathbb{F}_p}}(\log_p N) + O(N \log N)$, and the statement follows from the solution of this inequality. □

There are two challenges to find a faster polynomial multiplication algorithm over finite fields. The first challenge is the already mentioned existence of order sequences of constant sparseness over these fields. This conjecture is due to Bläser [5].

Conjecture (Bläser)**.** *There exist order sequences of constant sparseness over finite fields.*

In Remark 2 we showed, that indeed there exist suitable order sequences, however, they are too sparse for our purposes. The second challenge is the complexity of multiplication by powers of a primitive root of unity in extension fields. However, there are ways to overcome this with slight complexity increase. We recently obtained some progress in this area, and we think that a general improvement for fields of characteristic different from 2 and 0 is possible.

6 Conclusion

We generalized the notion of a DFT-based algorithm for polynomial multiplication, which describes uniformly all currently known fastest algorithms for polynomial multiplication over *arbitrary fields*. We parameterized fields by introducing the notion of the degree function and order sequences and showed upper and lower bounds for DFT-based algorithm in terms of these paremeters.

There is still an important open question whether one can improve the general Schönhage-Strassen's upper bound. As an outcome of this paper we support

the general experience that this question is not very easy. In particular, using only known DFT-based techniques will unlikely help much in case of arbitrary fields, in particular for the case of the rational field, as they did for the complexity of integer multiplication.

Acknowledgements

I would like to thank Markus Bläser for the problem setting and a lot of motivating discussions and to anonymous referees for many important improvement suggestions.

References

[1] S. Ballet. Low increasing tower of algebraic function fields and bilinear complexity of multiplication in any extension of $\mathbb{F}_q$. Finite Fields and Their Applications 9, pp. 472–478 (2003).

[2] S. Ballet, D. Le Brigand, and R. Rolland. On an application of the definition field descent of a tower of function fields. In Proceedings of the Conference Arithmetic, Geometry and Coding Theory (AGCT 2005), v. 21, pp. 187–203, Société Methématique de France, sér. Séminaires et Congrés, 2009.

[3] S. Ballet and J. Chaumine. On the bounds of the bilinear complexity of multiplication in some finite fields. Applicable Algebra in Engineering and Computing 15, pp. 205–211 (2004).

[4] S. Ballet and J. Pieltant. On the Tensor Rank of Multiplication in Any Extension of $\mathbb{F}_2$. arXiv:1003:1864v1 [math.AG] 9 Mar 2010.

[5] M. Bläser. Private communication.

[6] M. R. Brown and D. P. Dobkin. An improved lower bound on polynomial multiplication. IEEE Trans. Comput. 29, pp. 337–340 (1980).

[7] N. H. Bshouty and M. Kaminski. Multiplication of Polynomials over Finite Fields. SIAM J. Comput. 19(3), pp. 452-456 (1990).

[8] N. H. Bshouty and M. Kaminski. Polynomial multiplication over finite fields: from quadratic to straight-line complexity. Computational Complexity 15(3), pp. 252–262 (2006).

[9] P. Bürgisser, M. Clausen, and A. Shokrollahi. Algebraic Complexity Theory. Springer, Berlin, 1997.

[10] P. Bürgisser, M. Lotz. Lower bounds on the bounded coefficient complexity of bilinear maps. J. ACM 51(3), pp. 464–482 (2004).

[11] D. G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. Acta Informatica 28, pp. 693–701 (1991).

[12] D. Chudnovsky and G. Chudnovsky. Algebraic complexities and algebraic curves over finite fields. Journal of Complexity 4, pp. 285–316 (1988).

[13] M. Clausen and U. Baum. Fast Fourier Transforms. Wissenschaftsverlag, Mannheim-Leipzig-Wien-Zürich, 1993.

[14] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. Math. Comput. 19, pp. 297-301 (1965).

[15] A. De, P. P. Kurur, C. Saha, and R. Saptharishi. Fast integer multiplication using modular arithmetic. In Proceedings of the 40th ACM STOC 2008 conference, pp. 499–506.

[16] M. Fürer. Faster Integer Multiplication. In Proceedings of the 39th ACM STOC 2007 conference, pp. 57–66.

[17] M. Kaminski. An algorithm for polynomial multiplication that does not depend on the ring of constants. J. Algorithms 9, pp. 137–147 (1988).

[18] M. Kaminski. A Lower Bound On the Complexity Of Polynomial Multiplication Over Finite Fields. SIAM J. Comput. 34(4), pp. 960–992 (2005).

[19] M. Kaminski and N. H. Bshouty. Multiplicative Complexity of Polynomial Multiplication over Finite Fields. J. ACM 36(1), pp. 150–170 (1989).

[20] H. Hatalová and T. Šalát. Remarks on two results in the elementary theory of numbers. Acta Fac. Rer. Natur Univ. Comenian. Math. 20, pp. 113-117 (1969).

[21] V. Y. Pan. Simple Multivariate Polynomial Multiplication. J. Symbolic Computation 18, pp. 183–186 (1994).

[22] C. M. Rader. Discrete Fourier transforms when the number of data samples is prime. Proc. IEEE 56, pp. 1107-1108 (1968).

[23] A. Schönhage. Schnelle Multiplikation von Polynomen über Körpern der Charakteristic 2. Acta Informatica 7, pp. 395–398 (1977).

[24] A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. Computing 7, pp. 281–292 (1971).

[25] I. E. Shparlinski, M. A. Tsfasman, and S. G. Vladut. Curves with many points and multiplication in finite fields. Lecture Notes in Math. vol. 1518, Springer-Verlag, Berlin, pp. 145-169 (1992).